C28x Floating Point Unit

fastRTS Library

Module User's Guide

C28x Foundation Software

V1.00

SPRCA75

June 16, 2010

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgement, including those pertaining to warranty, patent infringement, and limitation of liability.

TI warrants performance of its semiconductor products to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, license, warranty or endorsement thereof.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations and notices. Representation or reproduction of this information with alteration voids all warranties provided for an associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible or liable for any such use.

Resale of TI's products or services with *statements different from or beyond the parameters* stated by TI for that products or service voids all express and any implied warranties for the associated TI product or service, is an unfair and deceptive business practice, and TI is not responsible nor liable for any such use.

Also see: Standard Terms and Conditions of Sale for Semiconductor Products.
www.ti.com/sc/docs/stdterms.htm

Mailing Address:
Texas Instruments
Post Office Box 655303
Dallas, Texas 75265

Contents

1. Introduction 4

2. Installing the Library 4

2.1. Where the Files are Located (Directory Structure) 4

2.2. Build options used to build the library 4

3. Using the fastRTS Library 5

3.1. Link Order of the Library 5

3.2. Header Files 9

3.3. Linker File 9

3.4. Confirming Which Library is Used 10

4. How to Rebuild the fastRTS Library 11

5. Function Summary 11

5.1. FPU fastRTS Function Summary 11

6. Function Descriptions 12

atan 12

atan2 12

cos 12

FS$$DIV 13

isqrt 13

sin 14

sincos 14

sqrt 14

7. Benchmarks 15

8. Included Example 16

9. Revision History 19

10. Legacy Information CCS v3.3 20

1.Introduction

The Texas Instruments TMS320C28x Floating Point Unit Fast Run-Time Support (RTS) library is a collection of optimized floating-point math functions for controllers with the C28x plus floating-point unit (FPU). This source code library includes C-callable optimized versions of selected floating-point math functions included in the compiler's standard run-time support libraries.

These routines are typically used in computationally intensive real-time applications where optimal execution speed is critical. By using these routines instead of the routines found in the existing run-time support libraries, you can achieve execution speeds considerably faster without rewriting existing code.

2.Installing the Library

2.1. Where the Files are Located (Directory Structure)

As installed, the *C28x FPU fastRTS Library* is partitioned into a well-defined directory structure. By default, the library and source code is installed into the following directory:

c:\tidcs\c28\C28x_FPU_fastRTS\<version>

Table 1 describes the contents of the main directories used by library:

Table 1. C28x fastRTS Library Directory Structure

Directory	Description
<base>	Base install directory. By default this is c:\tidcs\c28\C28x_FPU_fastRTS\v100 For the rest of this document <base> will be omitted from the directory names.
<base>\doc	Documentation including the revision history from the previous release.
<base>\lib	The built library.
<base>\include	Header file for non-standard functions such as isqrt() and sincos()
<base>\source	Source files for the library. This also includes a Code Composer Studio project that can be used to re-build the library if required.

2.2. Build options used to build the library

The 1.00 library is built with C28x codegen tools V5.0.2 with the following options:

-g -o3 -d"_DEBUG" -d"LARGE_MODEL" -ml -v28 --float_support=fpu32

3. Using the fastRTS Library

3.1. Link Order of the Library

To use the fastRTS functions in place of the existing functions the fastRTS library must be linked before the existing run-time support library. The fastRTS library replaces only a subset of the functions in the current run time support libraries. Therefore, the standard runtime support library should be linked after the fastRTS library.

The library can be used with CCS 3.3 or CCS 4. If you want to use the library within a CCS 3.3 project see the Legacy Information section at the end of this document. All examples provided with the library are Code Compser 4.x based.

1) Add the fastRTS and standard RTS libraries to the project using

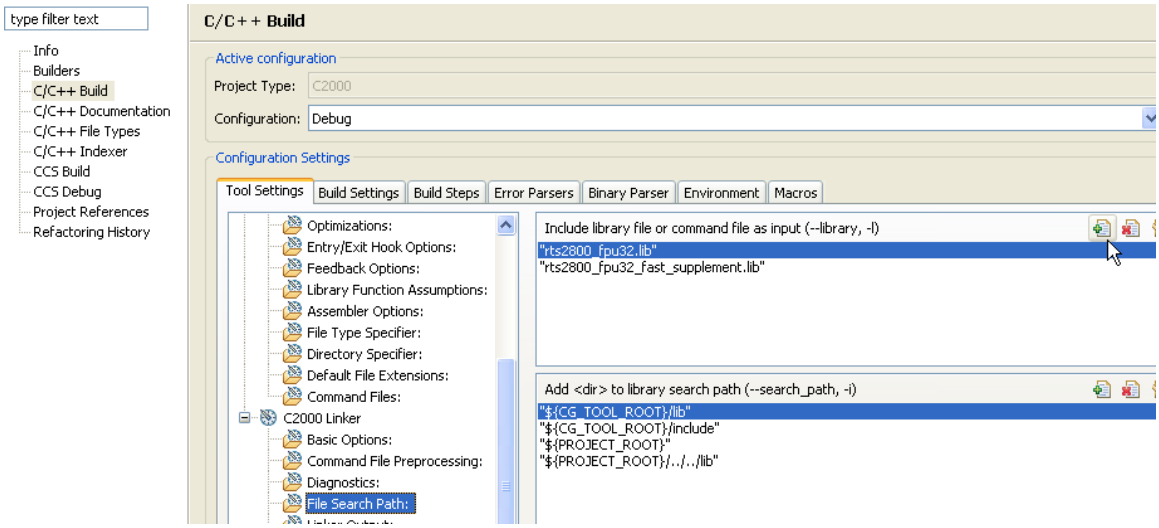
   a) Project->Properties or Right click on the project and select Properties

   b) Select C/C++ Build

   c) Under the C2000 Linker settings, Click on “File Search Path”

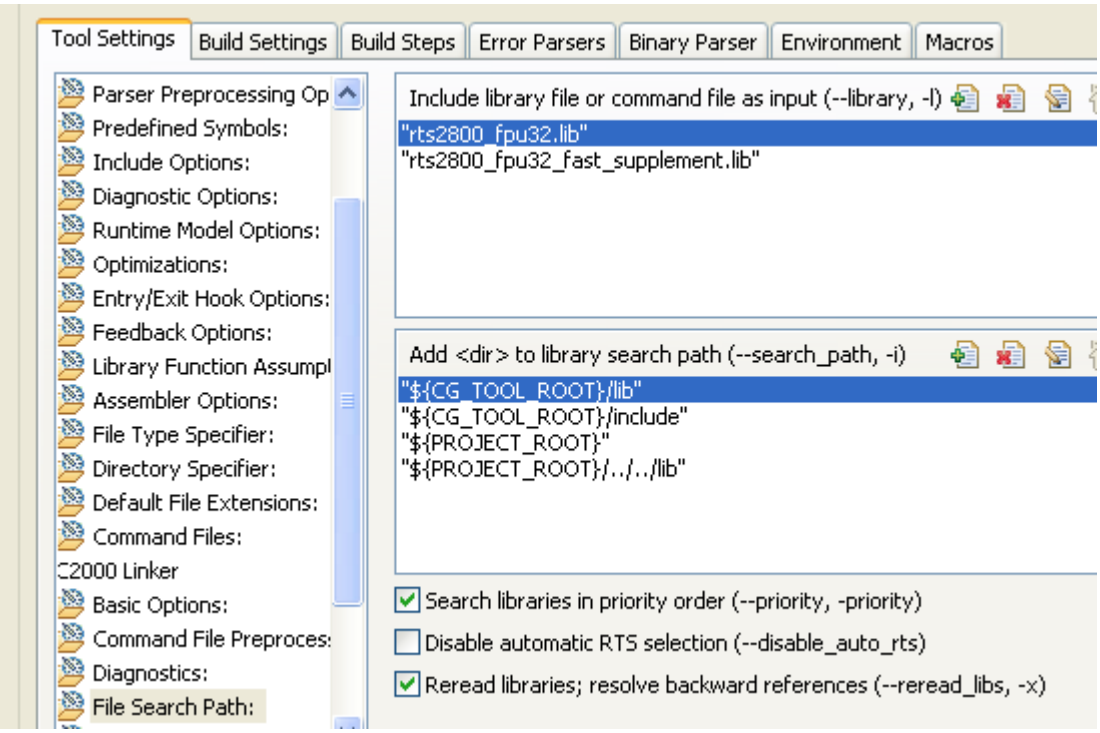
   d) In the “Include library file or command file” window, click on the + and add the fast RTS library: “rts2800_fpu_fast_supplement.lib”

   e) Make sure the path to the library is specified in the “Add <dir> to library search path” box. In the case shown below, the last entry points to the lib directory with the library.



2) On the File Search Path dialog box, make sure the following options are checked (at the bottom of the dialog box).

   a) Search libraries in priority order (-priority)
   b) Reread libraries; resolve backward references (-x)

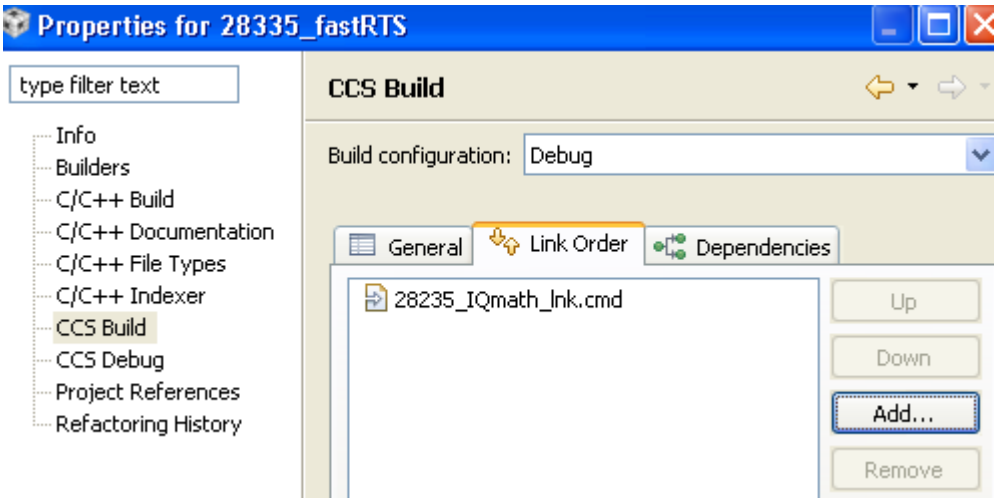


3) Specify the link order:

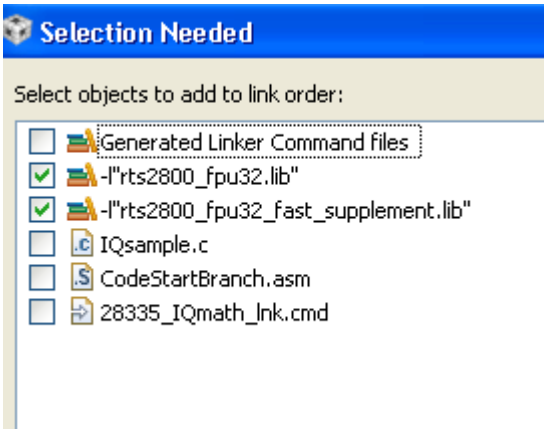
If the normal RTS library is first in the link order, then it will be searched first.
If the fastRTS library is first, then it will be searched first.

To use the fastRTS functions, make sure it is first in the link order by following these steps:

a) Click on the “CCS Build” Options and the “Link Order” tab.



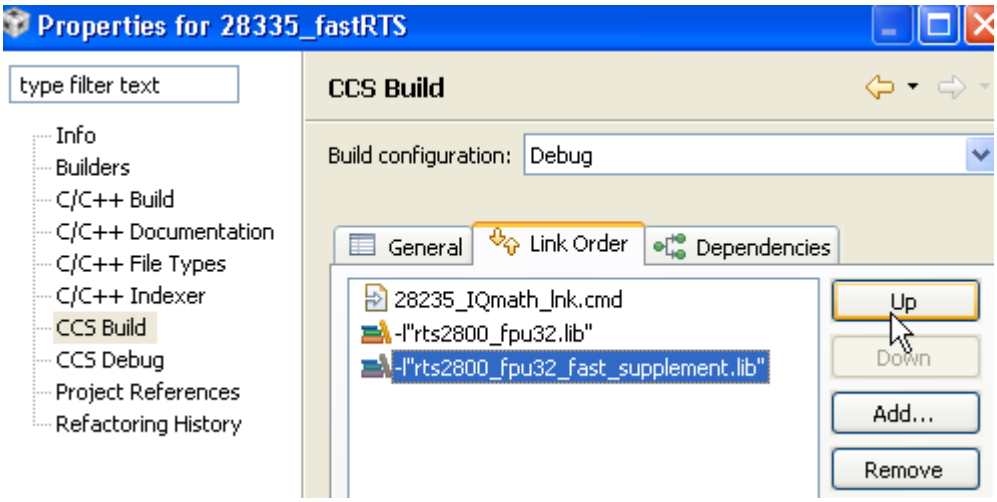
a) If both the rts2800 and fastRTS libraries are not listed, then click on the “Add” button and select them. Click “OK”



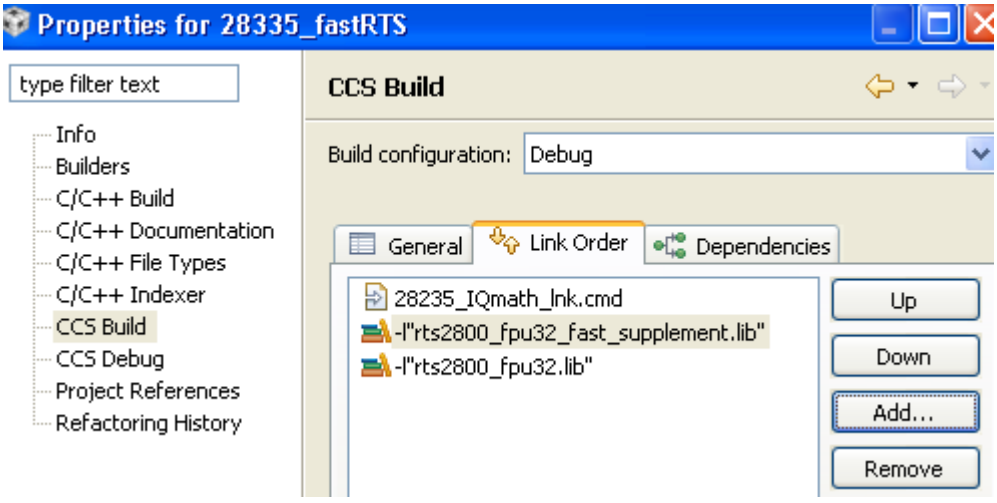
b) Move the fastRTS library up, so that it is linked in before the normal RTS library.

Click on the fastRTS library and then click on the “UP” button to move it up in the link order.

Normal RTS library (rts2800_fpu32.lib) will be searched first:



fastRTS library (rts2800_fpu_fast_supplement.lib) will be searched first:



3.2. Header Files

Use the same header files you would for the standard RTS library. For functions that are not part of the standard RTS library, use the included C28x_FPU_FastRTS.h header file.

.

3.3. Linker File

Many of the functions in the library use look-up tables to increase performance. These tables are located in the "FPUmathTables" memory section and are available in the boot ROM of the TMS320x2833x devices.

If you do not wish to load a copy of these tables into the device, use the boot ROM memory addresses and label the section as "NOLOAD" as shown below. This facilitates referencing the look-up tables without actually loading the section to the target.

Note that the boot ROM may not be zero-wait state on all devices and therefore using the boot ROM copy may add a few CPU cycles when compared to using the table loaded into SARAM. The impact to performance is minimal. Refer to the benchmarks section.

```
MEMORY
{
PAGE 0 :
   …
   FPUTABLES  : origin = 0x3FEBDC, length = 0x0006A0
   …
}
SECTIONS
{
   …
   FPUmathTables    : > FPUTABLES, PAGE = 0, TYPE = NOLOAD
   …
}
```

Note:

The addresses shown above are for the TMS320x2833x devices.

Note:

Using the fastRTS library may change the behavior of other standard RTS functions. For example, the fmod() function uses division. If the fastRTS library is used then the division portion will come from the FastRTS instead of the standard library.

3.4. Confirming Which Library is Used

After you build the project, check the .map file. This file is typically in the Debug directory of the project folder. This file will show which functions are being used from which library.

If the fastRTS library is linked in first, you will see something like the listing below. Notice the atan, cos, sin, sqrt and division functions are all coming from the fastRTS library.

```
text       0     00009000    000002a5
                 00009000      0000012e     fastRTS_sample.obj (.text)
                 0000912e      0000004a     rts2800_fpu32_fast_supplement.lib : atan_f32.obj (.text)
                 00009178      00000046     rts2800_fpu32.lib : boot.obj (.text)
                 000091be      00000034     rts2800_fpu32_fast_supplement.lib : cos_f32.obj (.text)
                 000091f2      00000034                                       : sin_f32.obj (.text)
                 00009226      00000021                                       : sqrt_f32.obj (.text)
                 00009247      0000001b     rts2800_fpu32.lib : args_main.obj (.text)
                 00009262      00000019                       : exit.obj (.text)
                 0000927b      00000019     rts2800_fpu32_fast_supplement.lib : div_f32.obj (.text)
                 00009294      00000009     rts2800_fpu32.lib : _lock.obj (.text)
```

If the normal RTS library is linked in first, you will see something like this. Notice the atan, cos, sin, sqrt and division functions are all coming from the normal RTS library library.

```
                 00009000      0000012e     fastRTS_sample.obj (.text)
                 0000912e      00000088     rts2800_fpu32.lib : fs_div.obj (.text)
                 000091b6      0000006f                       : atan.obj (.text)
                 00009225      0000005e                       : cos.obj (.text)
                 00009283      0000005c                       : sin.obj (.text)
                 000092df      00000046                       : boot.obj (.text)
                 00009325      00000001     --HOLE-- [fill = 0]
                 00009326      00000020                       : sqrt.obj (.text)
                 00009346      0000001b                       : args_main.obj (.text)
                 00009361      00000019                       : exit.obj (.text)
                 0000937a      00000009                       : _lock.obj (.text)
                 00009383      00000008     CodeStartBranch.obj (.text)
```

4. How to Rebuild the fastRTS Library

If you want to rebuild the fastRTS library (for example, because you modified the source contained in the archive), use the supplied Code Composer Studio project in the build_ccsv4 directory.

5. Function Summary

5.1. FPU fastRTS Function Summary

The following functions are included in this release of the fast RTS library. Other functions will be added in future releases. These functions are called as in the current runtime support library.

atan	isqrt
atan2	sin
cos	sincos
division	sqrt
	sincos

Note: isqrt() and sincos are not included in the standard RTS library.

6. Function Descriptions

atan	*Single-Precision Floating-Point ATAN (radians)*

Description Returns the arc tangent of a floating-point argument X. The return value is an angle in the range [-π, π] radians.

Header File
```
#include <math.h>
```

Declaration
```
float32 atan (float32 X)
```

atan2	*Single-Precision Floating-Point ATAN2 (radians)*

Description Returns the 4-quadrant arctangent of floating-point arguments X/Y. The return value is an angle in the range [-π, π] radians.

Header File
```
#include <math.h>
```

Declaration
```
float32 atan2 (float32 X, float32 Y)
```

cos	*Single-Precision Floating-Point COS (radians)*

Description Returns the cosine of a floating-point argument X (in radians) using table look-up and Taylor series expansion between the look-up table entries.

Header File
```
#include <math.h>
```

Declaration
```
float32 cos (float32 X)
```

FS$$DIV	*Single-Precision Floating-Point Division*

Description Replaces the single-precision division operation from the standard RTS library. This function uses a Newton-Raphson algorithm.

Header File None

Example

```
float32 X, Y, Z;
...
<Initialize X, Y>
...
Z = Y/X    // invokes FS$$DIV
```

Special Cases:

```
0.0/0.0 = +infinity
+FLT_MAX/+FLT_MAX =  0.0, LUF = 1
-FLT_MAX/+FLT_MAX = -0.0, LUF = 1
+FLT_MAX/-FLT_MAX =  0.0, LUF = 1
-FLT_MAX/-FLT_MAX = -0.0, LUF = 1
+FLT_MIN/+FLT_MAX =  0.0, LUF = 1
-FLT_MIN/+FLT_MAX = -0.0, LUF = 1
+FLT_MIN/-FLT_MAX =  0.0, LUF = 1
-FLT_MIN/-FLT_MAX = -0.0, LUF = 1
```

Division by 0.0 sets the LVF flag.

isqrt	*Single-Precision Floating-Point 1.0/Square Root*

Description Returns 1.0 /square root of a floating-point argument X using a Newton-Raphson algorithm.

Note: This function is not included in the standard RTS library. It is typically computed as 1.0L/sqrt(X). To use this function you must modify your code to instead call isqrt(X).

When migrating from an IQmath project, you can modify the IQmath header file to use isqrt(X) when configured for FLOAT_MATH.

Header File

```
#include "C28x_FPU_FastRTS.h"
```

Declaration

```
float32 sqrt (float32 X)
```

Special Cases isqrt(FLT_MAX) and isqrt(FLT_MIN) set the LUF flag.
isqrt(-FLT_MIN) will set both the LUF and LVF flags.
isqrt(0.0) sets the LVF flag.
If X is negative, isqrt(X) will set LVF and return 0.0.

sin	*Single-Precision Floating-Point SIN (radians)*

Description Returns the sine of a floating-point argument X (in radians) using table look-up and Taylor series expansion between the look-up table entries.

Header File
```
#include <math.h>
```

Declaration
```
float32 sin (float32 X)
```

sincos	*Single-Precision Floating-Point SIN and Cosine (radians)*

Description Returns both the sine and cosine of a floating-point argument X (in radians) using table look-up and Taylor series expansion between the look-up table entries.

Header File
```
#include "C28x_FPU_FastRTS.h"
```

Declaration
```
void sincos(float32 X, float32* PtrSin,
            float32* PtrCos);
```

X	Input argument in radians
PtrSin	Pointer to the sine result
PtrCos	Pointer to the cosine result

sqrt	*Single-Precision Floating-Point Square Root*

Description Returns the square root of a floating-point argument X using a Newton-Raphson algorithm.

Header File
```
#include <math.h>
```

Declaration
```
float32 sqrt (float32 X)
```

Special Cases
sqrt(FLT_MAX) and sqrt(FLT_MIN) set the LUF flag.
sqrt(-FLT_MIN) will set both the LUF and LVF flags.
sqrt(0.0) sets the LVF flag.
If X is negative, sqrt(X) will set LVF and return 0.0.

7. Benchmarks

The following table lists the execution time in CPU cycles for the fastRTS library routines. These numbers assume that both the code and stack are in zero wait-state memory. These numbers include function-call/return overhead but do not include any cycles for setting up the input data or storing the result.

Function	FPUmathTables in zero-wait SARAM	FPUmathTables in single-wait Boot ROM
atan	47	51
atan2	49	53
cos	38	42
division	24	24
isqrt	25	25
sin	37	41
sincos	44	50
sqrt	28	28

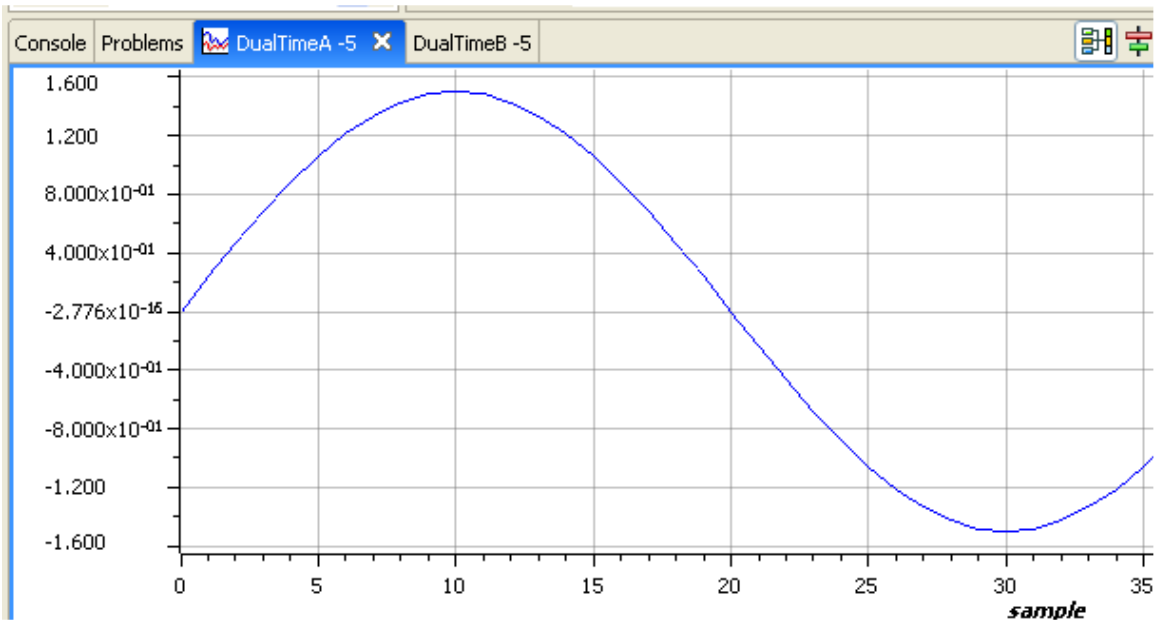
8. Included Example

In the examples_ccs4 directory there are projects for configured to use the normal RTS library and the fastRTS library. **The projects are identical except for the link order of the libraries (i.e. which library is searched first by the linker to resolve symbols.).**

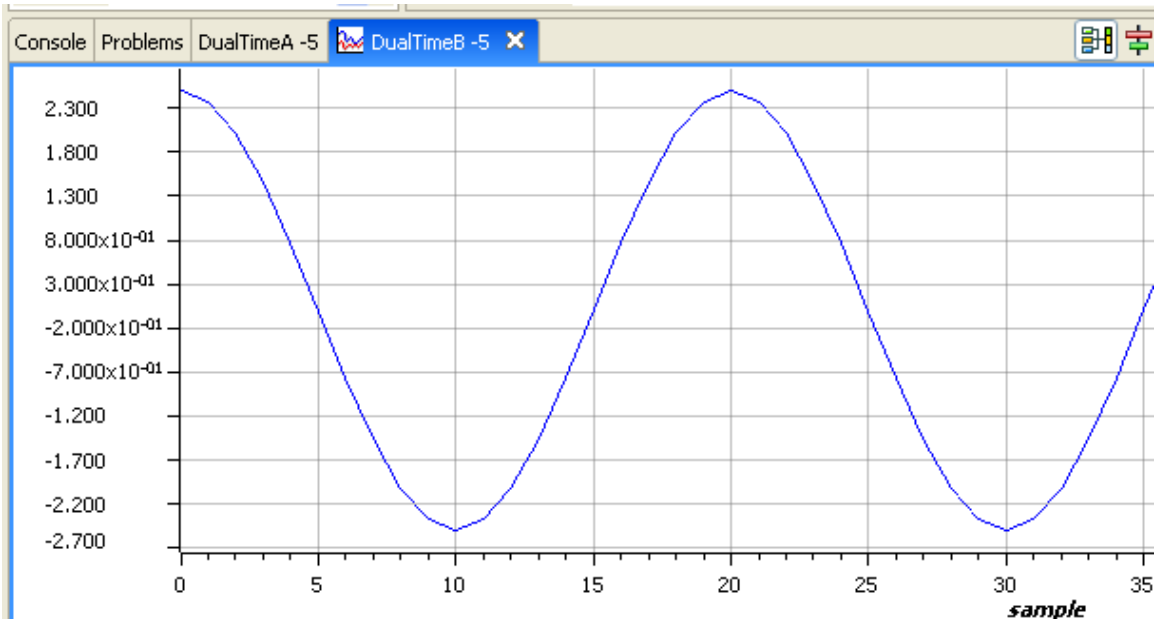
To view the waveforms in CCS 4, select tools->graphs-> dual time.
Then click on “import” and select the sin_cos.graphProp file in the directory of the example.
Click “OK”

Repeat these steps for atan_mag.graphProp.

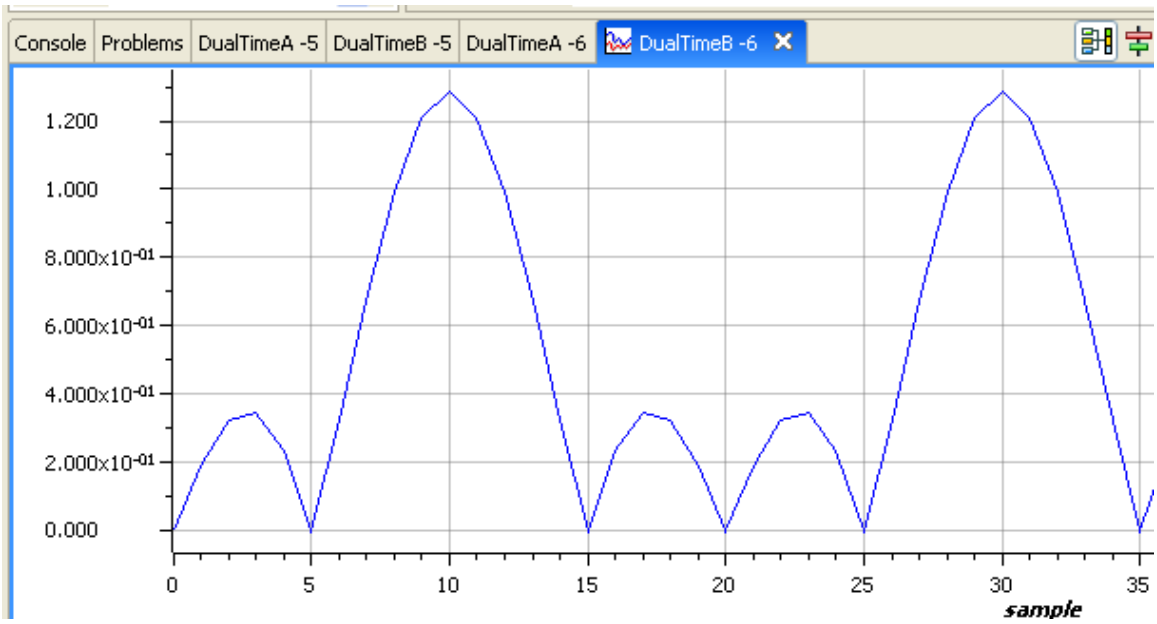
Sin Graph: Varaiable: Dlog.SINwaveform



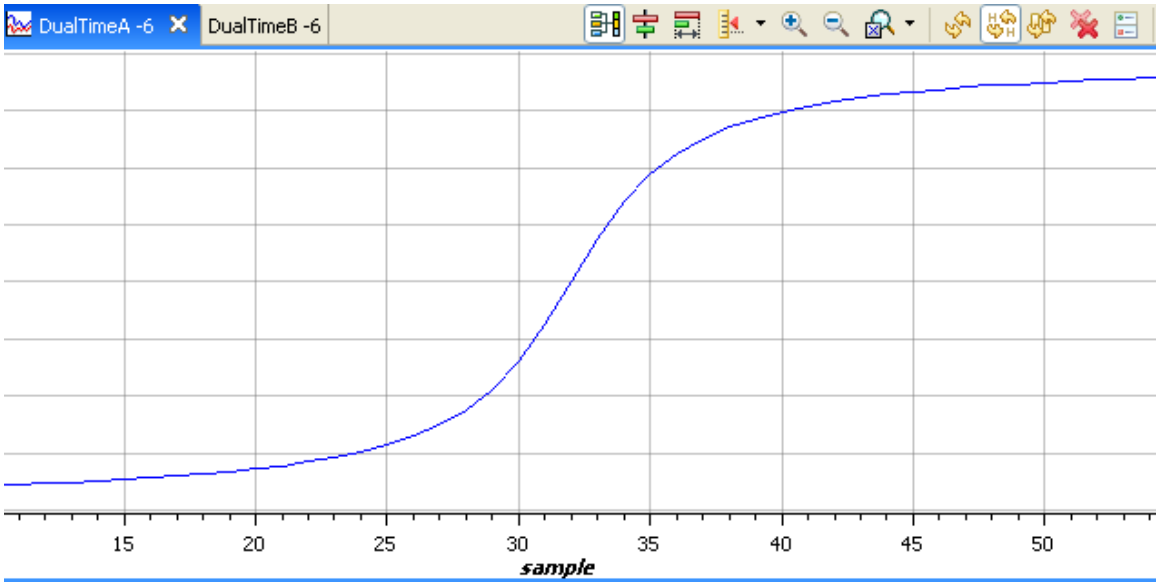
Cosine Graph: Varaiable: Dlog.COSwaveform



Mag Graph: Variable: Dlog.Mag



Mag Graph: Variable: Dlog.ATANwaveform



9. Revision History

Update for controlSUITE V1.00

No changes were made to the library itself. The following was done to incorporate the 1.00 release into the controlSUITE structure:

- Updated the directory structure to fit into controlSUITE
- Added a project to build the library using CCS 4.
- Added an example project using a CCS 4 based project
- Updated this document for CCS 4 and controlSUITE information.

Changes from Beta1 to V1.00

- Removed the version name from the library name. This makes updating to a new library easier.
- Added sincos() function
- Sin and Cos:
    - Corrected the constant value of 0.166 to 0.166667
    - Changed the truncated 2*pi/512 value to a rounded value of 2*pi/512. Previously this value was truncated.
    - In Beta 1, the int (Radian * 512/(2*pi)) calculation was done using float to 16-bit int. Changed this to 32-bit int to accommodate a larger range of input values.

10. Legacy Information CCS v3.3

Determining the link order for Code Composer Studio v3.3:

The .lib file can be used in either CCS 3.3 or CCS 4. This section describes how the link order is determined if the library is used within a CCS 3.3 project.

1. Add the fastRTS and standard RTS libraries to the project using

   Project->Add Files to Project

   Once the libraries are added to the project they will appear in the link order tab of the build options.

2. Open the build options dialog box under Project->Build Options

3. Under the Linker->Advanced tab, select the –priority linker switch.

   This will force the linker to resolve symbols to the first library linked.

4. Under the Link Order tab, select the two libraries and add them to the link order.

   Use the up/down arrows to arrange them in the proper order. The first library listed will be linked first.

5. Under the Linker->Libraries dialog add the path to the fast RTS library in the search path.

   Do not include either of the RTS libraries in the “Incl. Libraries” box. Doing so can cause problems when changing the link order since Code Composer uses both this field and the link order tab to determine which object files are linked first.

6. Save the project. (Project->Save).